Is Quantum Computing for Real?

An Interview with Catherine McGeoch of D-Wave Systems

by Walter Tichy

Editor's Introduction

In this interview, computer scientist Catherine McGeoch demystifies quantum computing and introduces us to a new world of computational thinking.

Is Quantum Computing for Real?

An Interview with Catherine McGeoch of D-Wave Systems

by Walter Tichy

Imagine the world's fastest computer requiring a totally new way of thinking: no sequencing, no conditions, no loops—indeed, no instructions. The machine is not a set of logic boxes controlled by clock; it is instead a large number of cells each containing a quantum bit or "qubit," connected with each other through weighted links. Changing just one bit triggers a cascade of changes in all the others. A qubit is no ordinary bit because it uses the quantum principle of superposition to store the two bit-values simultaneously. A link is no ordinary wire because it uses the quantum principle of entanglement to avoid relying on electric currents. In a machine with 2,000 qubits, all 2^{2000}possible combinations of bit-values are represented at the same time. When cooled to near absolute zero, and once a problem instance has been loaded into the qubits, the machine takes only a few microseconds to settle into a pattern that encodes the problem's solution. This, in short, is the computational model of quantum annealing and a way to build practical quantum computers. Quantum annealing might replace the Turing and Von Neumann models we're so used to and usher in a new kind of computational thinking.

That quantum effects such as superposition and entanglement could be used to build vastly more powerful computers was first suggested in the early 1980s courtesy of Richard Feynman of CalTech and David Deutsch of the University of Oxford. In 1994, Peter Shor, then at AT&T Bell Labs, showed how a quantum computer could be used to factor large numbers. The first quantum bits made of atoms, molecules, or photons appeared in the late 1990s and early 2000s. Despite the progress, physicists still struggle to construct more than a few qubits that can remain in superposition and entanglement for any useful time span.

Then in 1999, the Canadian company D-Wave Systems embarked on a mission to make quantum computing practical. Over the years, the company has been fabricating quantum computers with a steadily doubling number of qubits. In January 2017, D-Wave announced a machine with an unprecedented 2,000 qubits. With this many bits, realistic problems can be tackled.

In this interview, computer scientist Catherine McGeoch, who left Amherst College to join D-Wave Systems and specializes in NP-hard problems, will demystify quantum computing and introduce us to a new world of computational thinking.

Walter Tichy: Whenever a quantum computer in a research lab goes from, say, 8 qubits to 10, this is big news. Now D-Wave has announced a new quantum computer, going from 1,000 qubits to 2,000! What is going on here? Are we nearing practical quantum computing?

Catherine McGeoch: Most of those research labs are trying to build quantum computers that belong to the quantum gate model of computation (QGM). D-Wave implements a family of quantum annealing algorithms in hardware, which is the real-world counterpart of a different abstract model called adiabatic quantum computation (AQC) [1].

The engineering challenges that arise when trying to build working quantum computers under these two models of computation are considerably different. For QGM, it is a major technical challenge to minimize decoherence, which wrecks the probability of a successful computation (all quantum computation is probabilistic). That's why announcement of, say, a working 10-qubit device makes headlines—in the QG model it is a real achievement to manage decoherence at that scale.

The AQC model is more robust against decoherence (although coherence is still necessary). Annealing-based quantum computing has its own set of engineering challenges, but scaling up in qubit size while maintaining adequate levels of coherence hasn't really been on the top ten list.

As to whether we are nearing practical quantum computing, well, yes, that's the idea.

WT: Can you tell us which labs or companies have bought D-Wave computing systems, and what they are doing with them?

CM: Lockheed Martin has owned a D-Wave system since 2011, which it shares with USC. Their original D-Wave One (128 qubits) was upgraded to a D-Wave Two (500 qubits), and then a D-Wave 2X (1,000 qubits). Google, NASA, and Universities Space Research Association (USRA) have been sharing a D-Wave system since 2013. Their D-Wave Two was later upgraded to a D-Wave 2X, and they recently purchased an upgrade to the D-Wave 2000Q (2,000 qubits) system. Los Alamos National Laboratory acquired a D-Wave 2X system in 2015. Temporal Defense Systems purchased a D-Wave 2000Q system in early 2017.

There are also a number of customers who buy blocks of quantum computation time over the cloud, a service that has been available since 2010.

The best way to find out what they are doing with quantum computation is to visit their websites and look at the papers that are being published.

WT: I just noticed the first car company, Volkswagen, has announced a collaboration with D-Wave Systems to optimize traffic flow. Obviously, a D-Wave computer is not meant for your Hello World program or for surfing the Internet. Instead, quantum computers are supposed to tackle really tough problems. Before we go into concepts such as superposition and entanglement, which problems might readers understand that have been solved on D-Wave systems?

CM: The quantum processing unit (QPU) is designed to find good solutions to NP-hard optimization problems, which are defined in terms of certain cost functions to be minimized. These problems are found, for example, in operations research, artificial intelligence, and especially machine learning applications, which require large samples of solutions in their inner loop computations.

In principle, anything in NP—the famous class containing hundreds of challenging application problems—can be formulated for solution on the QPU. Of course, some inputs are too big to be solved directly on the QPU, in which case problem decomposition or a hybrid classical/quantum query approach may be used.

Note there are no theoretical guarantees available on how close to optimal the solution will be, or on how much time it would take to find an optimal solution. In this sense, quantum annealing is comparable to a classical optimization heuristic like simulated annealing.

The problems that have been actually implemented on D-Wave systems are too many to list here. Three recent papers describe problems in constraint satisfaction for circuit fault analysis [2], multiple query optimization [3] in databases, and constructing SAT filters (similar to Bloom filters) [4].

WT: **Can it crack codes, i.e., factor large numbers?**

CM: It can factor, but it does not use Shor's algorithm, it uses a different quantum annealing-based algorithm [5]. At current QPU sizes (which can handle numbers of size around 10 bits)

classical methods can beat certain lower-bound times imposed by the QPU control software. There are no theorems known about the efficiency of this algorithm, and we don't know if the quantum algorithm will be faster than classical methods when qubit counts grow large enough to solve more challenging inputs.

WT: What does the programming language look like?

CM: The QPU can compute any logic circuit, which technically means that is has the same capabilities as a classical ALU. But there isn't really a machine-level instruction set, there is just one (parameterized) instruction that basically says "solve it via quantum annealing."

Instead of figuring out how to write a program to solve your favorite problem, the intellectual challenge is how to formulate your problem so that it can be solved by this instruction. For example, the factoring problem is solved by formulating it as a multiplication circuit with inputs *X*, *Y* and output *N*, together with a cost function that is minimized when $N = X \cdot Y$. You set *N* to the number to be factored, and then ask the QPU to find *X* and *Y* to minimize the cost function.

This type of problem translation is a well-understood concept in NP-completeness theory, and cookbook translation methods are known for the optimization problems we are interested in solving. There is also a rapidly growing body of software tools available (C, Python, Fortran) to help the user formulate problems.

WT: How fast is it?

CM: Current-generation D-Wave systems can solve a 2000-variable problem with an anneal time of five microseconds. This is one of the minimum time bounds imposed by the control system: On some inputs the quantum algorithm could probably work fine with anneal times in tens of nanoseconds, and on other inputs longer anneal times give better results. The minimum elapsed time, including overhead for setup and readout to move the problem on and off the QPU, is about 10 milliseconds. So the quantum part of the computation is just a small fraction of elapsed time.

For optimization heuristics the question of "how fast" is usually evaluated in terms of the tradeoff between computation time and solution quality: How fast can it return a solution that meets a given quality threshold *X*? Or, for sampling applications: How fast can it return *N* distinct solutions of quality *Y*?

I don't have short answers to these questions, because they are highly input-dependent and hard to summarize. Serious performance evaluation of quantum annealing has only been possible in the last couple years. By comparison, scores of research groups have spent decades studying the (classical) simulated annealing heuristic, yet there is still no simple way to describe its performance.

WT: **One or two concrete examples, with solution qualities and runtimes, would help.**

CM: Here is an example from a recent paper: Trummer and Koch compared a D-Wave 2X system (1,000 qubits) to five classical algorithms on inputs for multiple query optimization, a database application [3]. They looked at the trade-off between computation time and solution quality. Generally speaking the QPU converged very quickly to very good solutions, whereas the classical solvers converged more slowly but could find comparable (sometimes better) solutions if given long enough time. With times (not counting setup) for the D-Wave system ranging from about one millisecond to one second, the software solvers took up to 1,000 times longer to "catch up" and return solutions of comparable quality. A separate test showed solutions found by the QPU in one second were typically within 0.4 percent of optimal.

At one end of the performance spectrum, Denchev et al. found a D-Wave 2X QPU can optimally solve a class of synthetic inputs 100 million times faster than some (but not all) classical algorithms [6]. This is a difference on the scale of one second versus three years of computation on a single core. At the other end, we know classical algorithms with nanosecond-scale instruction sets can solve small and/or easy inputs faster than the above-mentioned lower bounds. That is the current range of possibility that we are dealing with.

I should point out whatever numbers I give here—including those lower bounds—will likely improve as the technology continues to get better along with increasing qubit counts. Denchev et al., for example, noted their D-Wave 2X QPU solved the biggest problems 10,000 times faster than their predictions based on performance of the previous year's D-Wave Two QPU [6].

WT: **But can it beat a massively parallel computer?**

CM: Here is another question with no simple answers. First let me say the hoopla about quantum computation offering enormous speedups over classical computers is based on back-of-the envelope calculations from theoretical arguments about the worst-case cost of solving

NP-hard problems. That is, given an input with 2000 variables, a classical algorithm would have to search a space of 2^{2000} bit strings to certify it has found an optimal solution.

There have been about 2^{80} nanoseconds since the Big Bang, so solving a problem in five microseconds does indeed represent an unimaginably large speedup over that worst-case scenario. Parallel computation would not help unless you have an unimaginably large number of cores on hand (the number of atoms in the known universe is about 2^{240}). And in this theoretical context, you are allowed to have as many quantum computers as you have classical computers. In other words, parallel models of computation don't change the fundamental arguments about problem complexity.

However, the relevant theoretical questions are open and nobody knows if this line of reasoning is correct. Fast solution times for a given set of problems do not constitute a proof that all quantum computations are fast. Also, a faster classical algorithm might be discovered someday, thereby settling the famous $P = NP$ question in the affirmative.

No matter how the theory turns out, it is not really relevant to practice. For perhaps obvious reasons, nobody uses the worst-case approach to solve real application problems. Instead, classical heuristics are used, which can find pretty good solutions, pretty quickly, on some sets of inputs.

Having working quantum computers in hand makes it possible to compare performance against classical heuristics. The runtime of any such heuristic on an input of size 2,000 can be between a few microseconds (best case), up to Big Bang time scales (worst case). So the potential for enormous speedup is still there, but we don't know in advance which inputs will be easy or hard for which heuristics. This creates the "decades of research" issue I mentioned earlier.

The same issues apply when we talk about comparisons to massively parallel computing platforms: A back of the envelope calculation about theoretical speedup doesn't capture reality, and reality is hard to summarize. That being said, a couple of the heuristics we compare against have shown moderate runtime improvements from parallelization. In other cases, there is no speedup, and sometimes the parallel code is slower. Very generally speaking, it doesn't appear that parallel computation will be a huge difference-maker in this arena (a D-Wave technical report discusses this topic in more detail [7]).

WT: **What's inside the layers of cooling equipment of a D-Wave system? In other words, what are the bits and how are they linked?**

CM: It's not just cooling equipment. Like Schrödinger's cat, the QPU must do its work in a closed box, isolated as much as possible from ambient energy that could derail the quantum process. So it operates in a highly shielded environment where the temperature is below .015 kelvin, the magnetic field is 50 thousand times less than Earth's magnetic field, and atmospheric pressure is 10 billion times less than outside.

The qubits are made of microscopic loops of niobium, a metal that becomes a superconductor at temperatures below 9 kelvin. Qubits are linked together by couplers, also made of niobium. Figure 1 shows a photograph of a QPU chip inside its mounting apparatus—it is the black square in the center, about the size of a thumbnail.

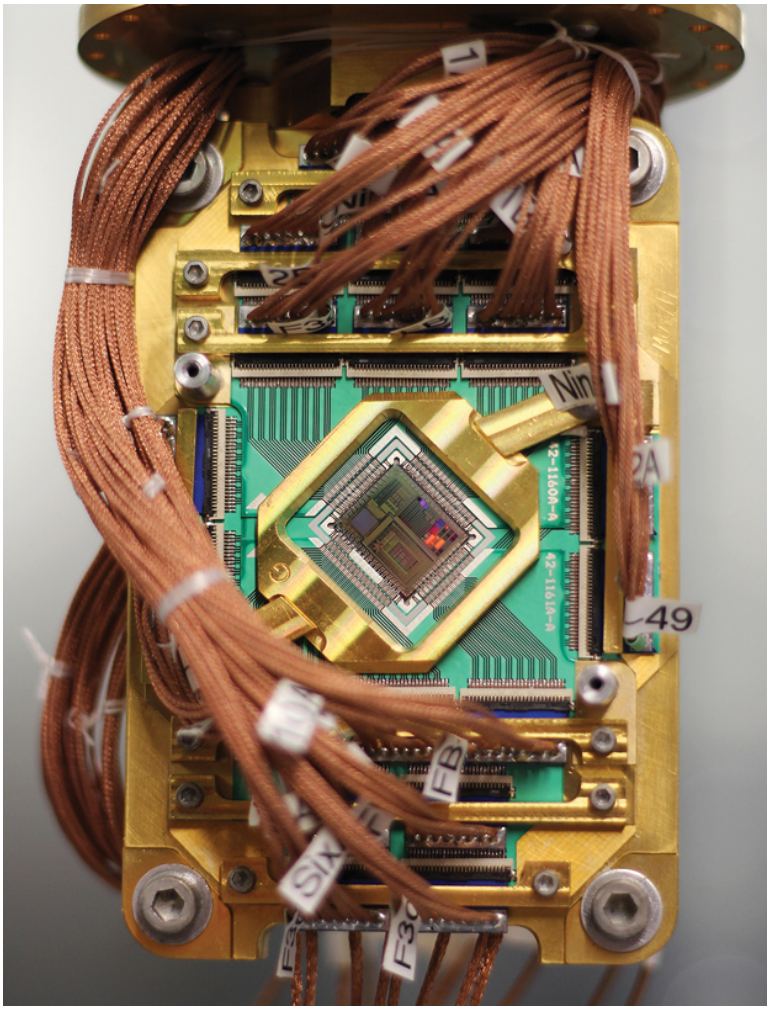


Figure 1. A quantum processing unit (QPU) in its mounting apparatus.

Figure 2 shows an 8x8 grid of so-called Chimera cells (on a smaller predecessor chip) containing the qubits and couplers that perform the actual computation. Each cell has eight qubits consisting of four thin loops across and four thin loops down, connected by couplers at their intersections. There are also couplers between qubits in one cell and those in its neighbor cells. Other components on the chip include qubit readout circuitry and a framework of digital-analog converters to control qubit and coupler states. A paper by Bunyk et al. describes some of the considerations that went into selecting this topology [8]. Basically, it offered the best combination of features within certain design constraints imposed by physics. (Other topologies are expected for use in future generations.)

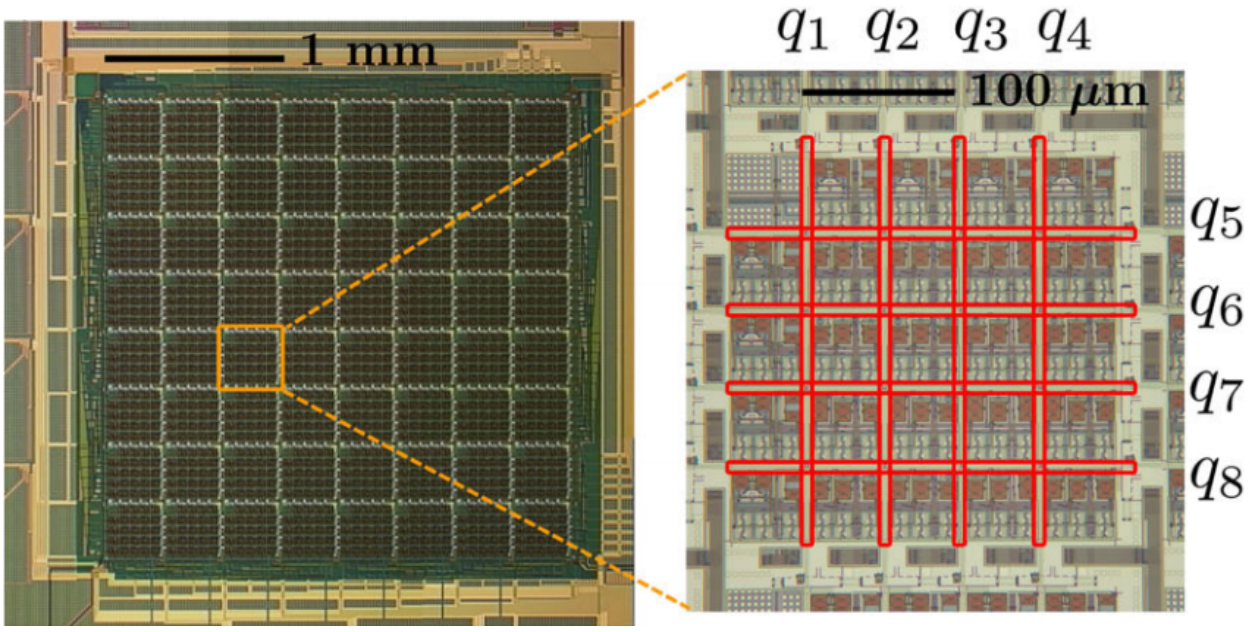


Figure 2. An 8x8 Chimera graph, with one Chimera cell magnified. The cell contains an arrangement of eight qubits, long thin loops of niobium marked in red, which are connected at their intersections by couplers. The interstices are filled with qubit read and control devices.

Figure 3 illustrates how this qubit arrangement corresponds to a Chimera graph. The left side shows eight qubits (red loops) and 16 couplers (blue ells). Also shown are eight couplers (blue circles) connecting to neighbor qubits in the north and east cells (couplers to the south and west are not shown here). The right side shows a graphical view where red nodes represent qubits and blue edges represent couplers. Each cell forms a 4x4 complete bipartite graph.

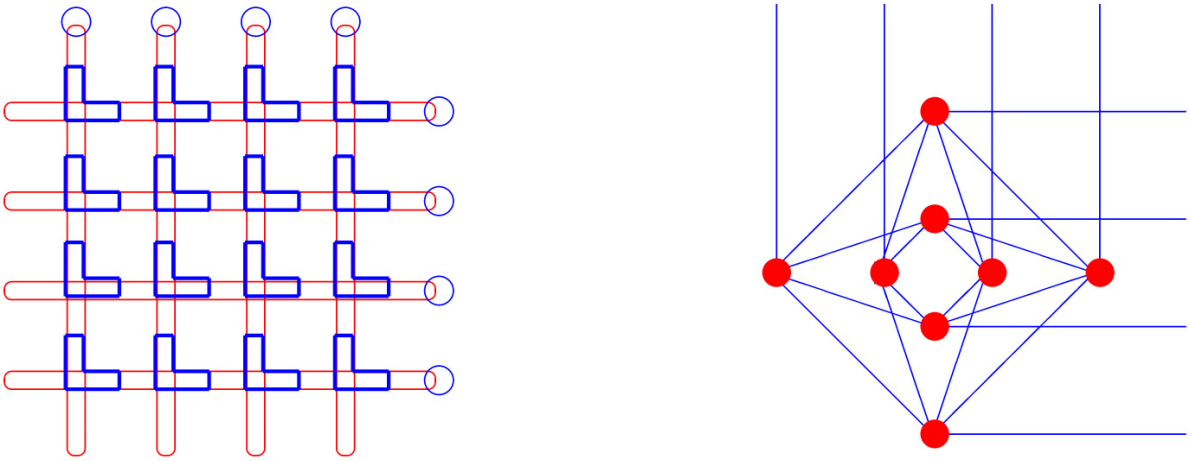


Figure 3. Left: A single Chimera cell contains eight qubits laid out in thin loops (red), four across and four down. Qubits are connected by couplers (blue cells) at their intersections. Also shown are eight of 16 possible couplings (blue circles) connecting to qubits in adjacent cells. Right: Qubit and coupler connectivity is described by a complete bipartite graph with eight nodes (red, representing qubits) and 16 edges (blue, representing couplers). Edges to qubits in neighbor cells located north and east are also shown.

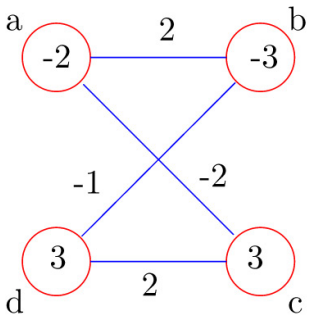
The 2000Q processor design is based on a 16x16 Chimera graph containing 2,048 nodes. The working graph for any given QPU is a subgraph of a Chimera graph, because an imperfect fabrication process leaves a small percentage of qubits and couplers unusable.

WT: **The fundamental problem the D-Wave processor solves is the Ising spin model. Please explain what that is.**

CM: You are given a graph $G = (V, E)$ with weights $h = \{h_i\}$ on the nodes and weights $J = \{J_{ij}\}$ on the edges. The problem is to assign spin values $s_i = \pm 1$ to the nodes so as to minimize the energy function defined by

$$E(s) = \sum_{i \in \mathcal{V}} h_i s_i + \sum_{(i,j) \in E} J_{ij} s_i s_j . \quad (1)$$

Figure 4 shows an example problem defined on a 4-node graph. The right side shows one possible spin assignment, which has energy $E(s) = -8$. This happens to be an optimal spin assignment for this input.



Spins:

$s_a : -1 \quad s_b : +1$

$s_c : -1 \quad s_d : +1$

$$E(s) = -2s_a - 3s_b + 3s_c + 3s_d + 2(s_a s_b) - 2(s_a s_c) - 1(s_b s_d) + 2(s_c s_d) = -8$$

Figure 4. An Ising model problem defined on a 4-node graph. The energy function is minimized at $E(s) = -8$ by the spin assignment shown here.

Ising spin model can be trivially transformed to quadratic unconstrained binary optimization (QUBO), defined on binary values $x = \{0, 1\}$ instead of spins, or maximum weighted two-satisfiability, defined on Boolean values $b = \{F, T\}$, which may be more familiar to computer scientists.

WT: Now for the hard part: How does the D-Wave computer solve this problem? Apparently, the machine itself is a graph. Edge and node weights will have to be loaded somehow into the graph, and the nodes must choose +1 or −1 so that $E(s)$ is minimized. But how does this happen?

CM: First, the Ising model input (h', J') defined on a general graph G is mapped to an equivalent problem (h, J) on the working graph C, using a process called minor-embedding (for examples, see [2, 3, 5, 9]). Roughly speaking, minor-embedding maps each node of degree d in G into $d/4$ nodes of degree 4 in C, plus additional edges to chain those nodes together. This process expands the total graph size somewhat, but the mapping is always sufficient to preserve the information in (h', J').

The h and J weights represent electromagnetic forces that push qubits towards certain magnetic polarities ($up = +1, down = -1$). For example, in the energy function (1), term $h_i s_i$ is minimized (i.e., negative) when h_i and s_i have opposite signs, and term $J_{ij}\ s_i\ s_j$ is minimized when J_{ij} is negative and s_i, s_j have matching signs, or J_{ij} is positive and s_i, s_j have opposite signs.

These forces and polarities correspond to various levels of magnetic energy, and the qubits naturally seek their lowest-energy state (called the ground state), just as water seeks the lowest point in a natural landscape.

A quantum annealing algorithm $H(r)$ is analog, and runs over a time interval $r: 0 \rightarrow 1$. It is specified by four components: (1) the *problem Hamiltonian* H_P, a matrix that describes system energies determined by h and J; (2) an *initial Hamiltonian* H_I that describes initial conditions; (3) a pair of functions $A(r)$ (decreasing in time) and $B(r)$ (increasing in time) that control a transition from H_I to H_P ; and (4) the total annealing time t_a for the transition, so $r = t / t_a$. Like this:

$$H(r) = A(r)H_I + B(r)H_P \text{ as } r = 0 \rightarrow 1. \qquad (2)$$

(See Wikipedia on quantum annealing for a description of the underlying physics.) At the end of the process, the qubit spin states are read and interpreted as a solution to the problem. Different quantum annealing algorithms correspond to different choices for these components. In the current 2000Q system, H_I is set to a default value, H_P and t_a are parameters set by the user, and $A(s), B(s)$ are from a family of functions that can be partially specified by the user.

WT: **Let me try to describe this process in my own words, since this is crucial. The initial Hamiltonian sets the qubit weights to neutral, meaning all the qubits can settle with equal probability on $+1$ or -1. The edge weights are also set to neutral, so the energy function $E(s)$ is the same for all possible combinations of $+1$ and -1 of the qubits. Now we gradually transition to the problem Hamiltonian by turning the weights up or down. The result is that $E(s)$ is no longer evenly distributed—we get an energy surface with peaks and valleys. While we do this, the annealing magic happens: the water pools in the valleys, meaning that the probabilities for settling in +1 or -1 shift in such a way as to prefer a minimum energy state. This happens over a time period in which the peaks and valleys become more pronounced. Is this picture correct?**

CM: Yes, that's about right, although I don't think "setting weights to neutral" is exactly the right way to put it. We have to think about it in terms of two quantum properties, superposition and entanglement.

Superposition. A classical bit can be in either state 0 or 1, and a register of N bits can only represent one of 2^N possible states at a time. But a qubit can be in superposition—simultaneously in both states at the same time, according to a certain probability. Superposition cannot be observed—like that famous cat, which is both alive and dead while the box is closed, but "snaps" to one state or the other when you open the box and look at it.

The initial Hamiltonian pushes qubits into an equal superposition of -1 and $+1$. For the full qubit system, this means that all possible outputs are equally likely.

At initialization time, $A(s)$ puts the initial Hamiltonian at full strength and $B(s)$ puts the problem Hamiltonian at zero strength. The transition gradually dials down $A(s)$ and dials up $B(s)$, so that the h and J forces are applied, first weakly and gradually becoming stronger. When the process finishes, H_I is at zero strength and H_P is at full strength. At the end, the qubit states are read and interpreted as a solution.

Appealing again to our water analogy, the water represents superposition probabilities. The superposition state can be everywhere on the surface at any time, covering all possible solutions, but it tends to pool in low-lying areas: A deeper pool at a given spot s corresponds to higher probability of s being read when the algorithm finishes.

This algorithm creates a surface that is initially flat, with water spread evenly across it. Then it gradually evolves from the flat surface to one that matches $E(s)$, and the probabilities for each qubit settle on either -1 or $+1$. If everything goes well (this is the probabilistic heuristic part), by the time the algorithm finishes, the water has collected in one spot that corresponds to an optimal solution.

WT: **Where does entanglement happen? I suppose you need it when splitting a problem node across several physical nodes, in which case they must maintain the same state. But if you link the split nodes with a strong negative weight, then the forces will make sure they have the same state. Is this entanglement? Some physicists claim there is no large-scale entanglement on D-Wave machines. How many nodes can a D-Wave system keep entangled?**

CM: A group of entangled qubits can change their (superposition) states simultaneously in an indivisible way, rather than individually. The edge weights create magnetic couplings between

qubits: these couplings create entanglement in the ground state of the qubit system as it evolves.

Consider a small 2-qubit system with four possible spin configurations, with weights $h_0, h_1 = 0$ and $J_{01} = +1$. The classical minimum-energy spin states are $(+1, -1)$ and $(-1, +1)$. During the anneal, the ground state for the system is a superposition of both states. So the individual qubits are neither +1 nor -1, they are in equal superposition, pointing up and down at the same time. Due to entanglement, their superposition states are linked, so that, if you read one and then the other they would have opposite signs. This mysterious instantaneous linkage is not the same as a magnetic signal, which would apply to classical states (up or down only), and would attenuate over distance.

In the context of quantum annealing, superposition and entanglement together create a phenomenon known as "tunneling." In our water metaphor, this means under certain circumstances the landscape is effectively porous, and the qubit states can go directly through hills rather than climbing over them. Classical algorithms (and magnets) cannot tunnel: They must change state by climbing over hills and moving around the landscape step by step.

Tunneling is the magical property that, in an ideal scenario, allows the quantum algorithm to slide directly to a ground state even if there are hills along the way, as long as the anneal time t_a is above a certain threshold (unknown in practice). No real-world quantum computation can achieve the theoretically ideal situation, though. Despite all the shielding and cold temperatures, there is always some amount of noise from the ambient environment that interferes with the computation. We have a quantum annealing algorithm that uses tunneling in limited ways, and we are working to understand this phenomenon.

As far as claims that there is no large-scale entanglement, I think it would be more accurate to say large-scale entanglement has not been demonstrated. Indeed, it is a significant scientific challenge to perform an experiment to detect entanglement—which only takes place when the qubits are shielded from ambient energy—since energy from the experimental apparatus can destroy the very effect it is trying to detect. The physics experiments demonstrating entanglement in a D-Wave QPU were only designed to detect entanglement in an 8-qubit system [9].

A later non-intrusive experiment based on analyzing outputs from carefully-constructed inputs, showed tunneling does occur (which could only happen if entanglement is present) on systems of 945 qubits [6]. But I don't think it is possible to infer from that result how many qubits actually did tunnel, or how many were entangled.

WT: **You mentioned that quantum annealing is analog. I suppose the edge and node weights and the forces are analog. What is the resolution of these weights, i.e., how many possible values are there? And how does the analog computation affect the problems that can be solved and the answers generated?**

CM: Right, the transition of forces from H_I to H_P is analog. The weights can be specified as regular floating-point numbers. Analog control errors create small perturbations on h and J, so the energy function that the QPU solves is slightly different from the one specified by the user. If the difference is big enough that states swap ranks (i.e., the best solution in the original problem is only second-best in the perturbed problem), then the algorithm has lower probability of finishing in ground state, and higher probability of finishing in the next-best state.

I can't give an exact number for the required weight resolution on inputs of interesting size: it depends on how far apart the energies of the low-energy states are. Since we work with NP-hard problems we don't typically have knowledge of the optimal and near-optimal states, so this quantity is hard to analyze or predict in practical scenarios.

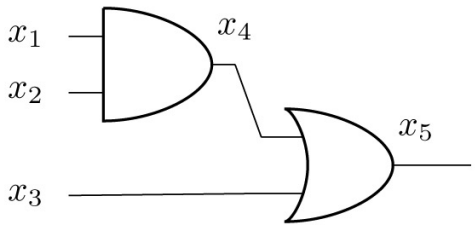
There are various ways to compensate for analog errors; for example, a system utility is available that can average-out systematic errors, similar to the way randomized quicksort avoids the worst-case scenario. Also, H_P can be re-formulated, or techniques of quantum error correction can be applied, to increase the separation between ground states and other states in $E(s)$, to mitigate the rank-swapping scenario. Also, as mentioned previously, new generations of processors tend to show improved performance due to better suppression of these types of errors.

WT: **Does quantum annealing require quantum entanglement?**

CM: Quantum annealing exploits both quantum entanglement and superposition, but those properties are not necessarily required to solve every input. For example, if the landscape is shaped like a big cherry-bowl instead of having lots of hills and valleys, then the quantum algorithm does not need to invoke quantum tunneling: It just slides down the hill like a classical greedy algorithm would. Entanglement would be present but simple magnetic forces would be enough to solve this problem.

WT: **Now that we have an idea of how the machine works, please explain how a concrete problem is mapped to the Ising model, so the machine can solve it.**

CM: In the circuit satisfiability problem, you are given a binary circuit and the question is whether there is an assignment of values to inputs that makes the circuit evaluate to 1 (assume that $0 = F$ and $1 = T$). Figure 5 shows a small example circuit with inputs x_1, x_2, x_3 and two gates, AND above and OR below. This circuit has value 1, for example, when $x_1 = 1$, $x_2 = 1$ and $x_3 = 0$.



Clauses:		IM Expressions:
$(x_5 \text{ EQ } 1)$	$\Rightarrow$	$-3s_5$
$(x_5 \text{ EQ } (x_4 \text{ OR } x_3))$	$\Rightarrow$	$s_3 + s_4 - 2s_5 + s_3s_4 - 2s_3s_5 - 2s_4s_5$
$(x_4 \text{ EQ } (x_1 \text{ AND } x_2))$	$\Rightarrow$	$-s_1 - s_2 + 2s_4 + s_1s_2 - 2s_1s_4 - 2s_2s_4$

Figure 5. Transformation from circuit satisfiability to Ising model. Each gate in the circuit corresponds to a Boolean clause, plus an extra clause (top row) that says the circuit output is 1. Each clause is translated to an arithmetic expression defined on variables $s_i \in \{-1, +1\}$. The circuit inputs x_1, x_2, x_3 (with values $\{0, 1\}$) make the circuit evaluate to 1 ($True$) exactly when the corresponding expression inputs s_1, s_2, s_3 (with values $\{-1, +1\}$) minimize the sum of these three expressions.

Step one in the transformation is to write each gate as a Boolean clause of the form $(z\ EQ\ (x\ OP\ y))$, where OP is the gate operation and the clause is true when the output wire z equals the result of the operation applied to input wires x and y. The two clauses for our example circuit are listed below it, together with an extra clause on top that says the circuit output is equal to 1. For inputs x_1, x_2, x_3 the circuit value is 1 exactly when all three clauses are satisfied, that is, all three have value 1.

The next step is to transform each clause to an arithmetic expression. Assume the binary values (0,1) map to spin values $(-1, +1)$, respectively. Since Ising model is a minimization problem,

the goal is to build an energy function $E(s)$ defined on $(-1, +1)$ that has minimum energy exactly when the corresponding binary values (0,1) satisfy all three clauses. Three expressions matching the three clauses are shown at the bottom right of Figure 5. (There are different ways to set up these expressions; see references for more information [2, 9].)

Some details of this correspondence for the OR clause in the middle row are shown in Figure 6. The left side of the truth table shows all possible inputs and outputs for the clause defined on x_3, x_4, x_5: The clause value is 1 in rows where x_5 equals the OR of x_3 and x_4. The right side shows the corresponding inputs and outputs for the Ising mode expression defined on s_3, s_4, s_5. In every row where the clause is satisfied, the expression is minimized with value -3.

x_3 x_4 x_5	OR clause	s_4	s_3	s_5	IM expression
0 0 0	1	-1	-1	-1	-3
0 0 1	0	-1	-1	$+1$	1
0 1 0	0	-1	$+1$	-1	1
0 1 1	1	-1	$+1$	$+1$	-3
1 0 0	0	$+1$	-1	-1	1
1 0 1	1	$+1$	-1	$+1$	-3
1 1 0	0	$+1$	$+1$	-1	9
1 1 1	1	$+1$	$+1$	$+1$	-3

Figure 6. The clause truth table (left) and the Ising model expression (right) show all possible inputs and outputs for these two versions of the OR circuit of Figure 5. The clause is satisfied (has value $1 = True$) in the same rows for which the IM expression is minimized (has value -3).

The requirement that all clauses must be satisfied becomes a requirement that the sum of all expressions must be minimized. The last step in the transformation is to add the expressions together and collect terms (constants can be discarded). The result is an Ising model input as shown on the left of Figure 7. One possible way to map (h, J) to a Chimera cell is to use the labeling scheme shown on the right (the weights of unused nodes and edges are set to 0).

Input	Value	Input	Value
h_1	-1	J_{12}	1
$h_2 = h_{2a} + h_{2b}$	-1	J_{14}	-2
h_3	1	J_{24}	-2
h_4	3	J_{35}	-2
h_5	-5	J_{45}	-2
		J_{2a2b}	-10

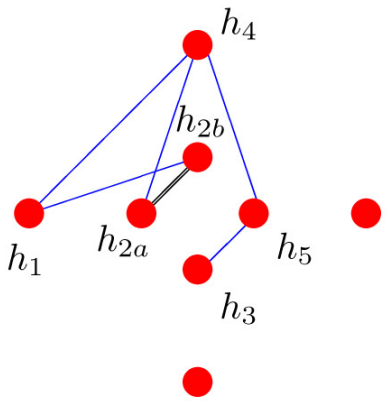


Figure 7. Left: Ising model inputs (h, J) for the circuit of Figure 4. Right: A mapping of (h, J) onto a Chimera cell. Nodes with $h = 0$ are not labeled, edges with $J_{ij} = 0$ are not shown. The h_2 weight is split across two nodes, which are connected by a chain (black edge) having a strong negative weight, in this case -10.

Note h_2 is split across two nodes that are connected by a so-called "chain" (black edge), which is set to a strong negative value like -10 to ensure they end up with the same spin in the output. This split is necessary because the triangle J_{12}, J_{14}, J_{24} can't be embedded directly onto the Chimera topology; this decision is part of the minor-embedding process mentioned earlier.

Send this problem to the QPU, and it will return a sample of spin assignments $s_1 \ldots s_5$, some of which correspond to ground states of (h, J) (there are four such ground states). In those states, the assignments of spin values to s_1, s_2, s_3 can be mapped back to assignments of binary values to x_1, x_2, x_3 for which the circuit value is 1, using our convention that spin -1 equals binary 0 and spin $+1$ equals binary 1.

WT: **Thank you, Catherine, for this very informative interview. Obviously, a QPU is very different from a traditional CPU. There is only a single solver hardwired into the machine, such that the qubits all tug on each other in order to find a solution to the Ising model. The D-Wave system seems to be a very cool (literally) and versatile accelerator for combinatorial problems.**

This interview has been condensed and edited for clarity.

For those who want to see the machine, a 3-part tour of the D-Wave system can be found on YouTube: look for "D-Wave lab tour." Catherine's monograph, aimed at a computer science audience, discusses adiabatic quantum computation and quantum annealing in more detail [11]. Catherine would like to thank Andrew Berkley and Mark Johnson of D-Wave for helping out with the physics discussion.

References

[1] Farhi et al. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science* 292, 5516 (2001).

[2] Z. Bian et al. Mapping constrained optimization problems to quantum annealing with applications to fault diagnosis. *Frontiers in ICT* 3, 14 (2016).

[3] Trummer, I. and Koch. C. Multiple query optimization on the D-Wave 2X adiabatic quantum computer. *Proceedings of the VLDB Endowment* 9, 9 (2016).

[4] A. Douglass et al. Constructing SAT Filters with a Quantum Annealer. *International Conference on Theory and Applications of Satisfiability Testing—SAT 2015.* Springer LNCS 9340, 2015, 104-120,.

[5] E. Andriyash et al. Boosting integer factoring performance via quantum annealing offsets. D-Wave Technical Report 14-1002A-B. 2016. Available at http://www.dwavesys.com/resources/publications.

[6] Denchev et al. What is the computational value of finite range tunneling?, *Physical Review X* 6, 3 (2015.)

[7] Limits on Parallel Speedup for Classical Ising Model Solvers. D-Wave White Paper 14-1004A-A. 2017; https://www.dwavesys.com/sites/default/files/14-1004A_wp_Limits_on_Parallel_Speedup_for_Classical_Ising_Model_Solvers.pdf

[8] Bunyk et al. Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Transactions on Applied Superconductivity* 24, 4 (August 2014).

[9] Su et al. A quantum annealing approach for Boolean satisfiability problem. In *Proceedings of the 53rd Annual Design Automation Conference* (DAC'16). Article 148. ACM, New York, 2016.

[10] Lanting et al. Entanglement in a quantum annealing processor. *Physics Review X* 4, 021041 (May 2014).

[11] McGeoch, C. C. *Adiabatic Quantum Computation and Quantum Annealing*. Morgan & Claypool Publishers, 2014.

About the Author

Walter Tichy has been professor of Computer Science at Karlsruhe Institute of Technology (formerly University Karlsruhe), Germany, since 1986. His major interests are software engineering and parallel computing. You can read more about him at www.ipd.uka.de/Tichy.

DOI: 10.1145/3084688

